DumBO

February 15, 2023

[1]:
```
%matplotlib inline

import time

import numpy as np
import matplotlib.pyplot as plt
from IPython.display import clear_output

import dumbo
```

1 DumBO

Teach the basic concepts of BO by building a simplistic optimizer.

We want to find the value of `x` that maximizes the function, `y(x)`, as defined by `measure()`, below. This may be called *objective function*.

[2]:
```
def measure(x):
    return float(1 - .0950 + (
        -(((x - .3333)**2).mean())
        + .1*np.sin(30*x).mean()
        + .01*np.random.normal()
    ))
```

In the Bayesian optimization setting it is *expensive* – in dollars, time, risk, labor, etc. – to measure `y(x)` as a value `x`. Therefore, we try to take as few measurements as possible.

Typically measurements are uncertain – they're *noisy*. We simulate measurement noise in `measure()` by adding `.01*np.random.normal()` to the function value.

Note, also, that we don't know the derivative of `y(x)`. We only know measured function values. This property makes the problem a *black-box* optimization problem.

In summary, Bayesian optimization is said to perform black-box optimization of noisy, expensive objectives.

[3]:
```
def surrogate(x_m, y_m, x):
    distance = np.sqrt( ((x - x_m)**2) )
    i = np.argmin(distance)
```

```python
    y_ex = y_m[i]
    y_var = distance[i]
    return y_ex, y_var
```

[4]:
```python
def acquisition_function(y_ex, y_var):
    return y_ex + np.sqrt(y_var)/2
```

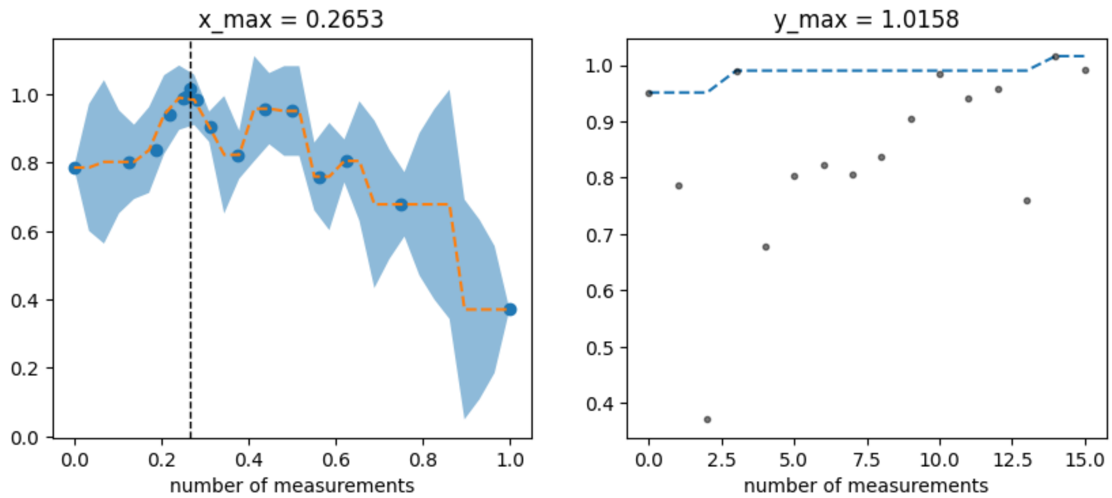
[5]:
```python
def optimize(x_m, y_m):
    x = np.linspace(0,1,1000)
    af = np.array([acquisition_function(*surrogate(x_m, y_m, xx)) for xx in x])
    i = np.argmax(af)
    return x[i]
```

[6]:
```python
def plot_surrogate(ax, x_m, y_m):
    x = np.linspace(0,1,30)
    ev = np.array([surrogate(x_m, y_m, xx) for xx in x])
    y_ex = ev[:,0]
    y_se = np.sqrt(ev[:,1])

    ax.plot(x_m, y_m, 'o');
    ax.plot(x, y_ex, '--');

    ax.fill_between(
        x,
        y_ex - y_se,
        y_ex + y_se,
        alpha=.5,
    linewidth=1
    );

def vline(ax, x0, color='black'):
    c = ax.axis()
    ax.autoscale(False)
    ax.plot([x0, x0], [c[2], c[3]], '--', linewidth=1, color=color)
```

[ ]:
```python

```

[7]:
```python
phi = ( 1 + np.sqrt(5) ) / 2 # the golden ratio, for plots
y_best = 0
x_m = np.array([.5])
y_m = np.array([measure(xx) for xx in x_m])

trace = [y_m[0]]
for _ in range(15):
    x = optimize(x_m, y_m)
    y = measure(x)
    x_m = np.append(x_m, x)
```

```
    y_m = np.append(y_m, y)
    trace.append(y_m.max())

    xlabel = 'number of measurements'
    width = 10
    clear_output(wait=True)
    fig, (ax1, ax2) = plt.subplots(1,2, figsize=(width, width/phi/phi))
    plot_surrogate(ax1, x_m, y_m)
    x_max = x_m[np.argmax(y_m)]
    ax1.set_title(f"x_max = {x_max:.4f}")
    vline(ax1, x_max)
    ax1.set_xlabel(xlabel)
    ax2.plot(trace, '--');
    ax2.plot(y_m, '.k', alpha=.5);
    ax2.set_title(f'y_max = {y_m.max():.4f}')
    ax2.set_xlabel(xlabel)

    plt.show()

    time.sleep(.05)
```



[8]:
```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
import xgboost as xgb

np.random.seed(17)
digits = load_digits()

n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))
```

```
x_train, x_test, y_train, y_test = train_test_split(
    data, digits.target, test_size=1/2, shuffle=True
)
```

[9]:

```
def measure_test_accuracy(x):
    x = x.flatten()
    hp = {
        'max_depth': int(1 + 7*x[0] + .5),
        'subsample': x[1],
        'min_child_weight': 1 + 99*x[2],
        'colsample_bytree': x[3],
        'eta': x[4],
        'num_parallel_tree': int(1 + 9*x[5] + .5),
    }
    xgb_model = xgb.XGBClassifier(
        objective="binary:logistic",
        **hp
    )
    xgb_model.fit(x_train, y_train)
    return xgb_model.score(x_test, y_test)
```

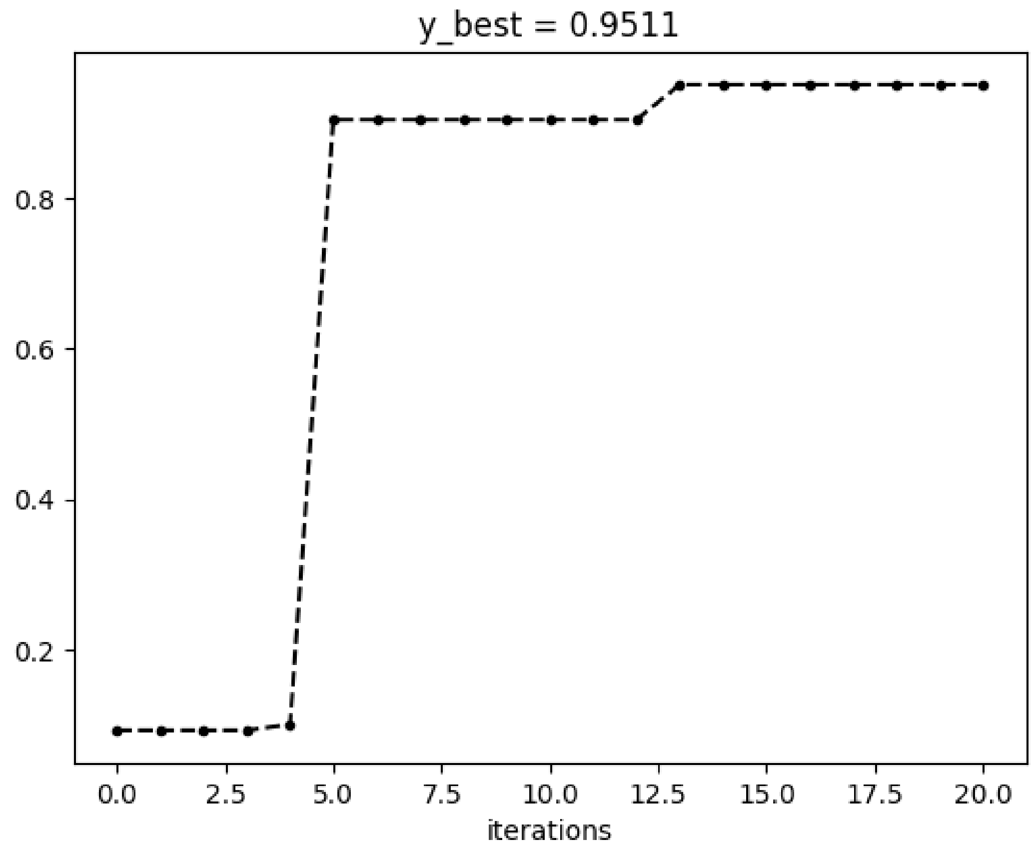
[10]:

```
num_dim = 6
x_m = np.random.uniform(size=(1, num_dim))
y_m = np.array([measure_test_accuracy(x_m)])

y_best = y_m[0]
x_best = x_m[0]
trace = [y_best]
print (y_best)
```

0.09232480533926585

[12]:

```
for _ in range(20):
    x = dumbo.optimize(x_m, y_m)
    y = measure_test_accuracy(x)
    x_m = np.append(x_m, x, axis=0)
    y_m = np.append(y_m, y)
    if y > y_best:
        y_best = y
        x_best = x
    trace.append(y_best)
    clear_output(wait=True)
    plt.plot(trace, 'k.--');
    plt.title(f"y_best = {y_best:.4f}")
    plt.xlabel('iterations')
    plt.show()
```


y_best = 0.9511
0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
iterations

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: